INF404 - Travaux pratiques - (semaines 1 et 2)

TP1 : Analyse et evaluation d'Expressions Arithmétiques "Simples" (EAS)

Page moodle du cours : `https://im2ag-moodle.univ-grenoble-alpes.fr/course/view.php?id=385`

Avant de commencer cette séance :

1. **Inscrivez-vous** sur Moodle dans votre groupe de TP (si ce n'est pas déjà fait ...)
2. créez un répertoire ***INF404***, puis un répertoire ***TP1*** dans ce répertoire
3. placez-vous dans votre répertoire ***INF404/TP1***
4. récupérez les fichiers nécessaires à ce TP sur moodle et placez-les dans ce répertoire.

Rappel (à la fin de chaque TP) :

— créez une archive (au format tar, zip, etc.) contenant les fichiers produits dans la séance
— déposez cette archive sur Moodle dans la zone de dépôt correspondant à votre groupe de TP

Exercice 1 - comprendre les programmes fournis

Examinez les fichiers ***lecture_caracteres.h***, ***lecture_caracteres.c***, ***analyse_lexicale.h*** et ***analyse_lexicale.c***.

Dans ce dernier fichier examinez particulièrement la procédure `reconnaitre_lexeme` qui "produit" un nouveau lexème en se basant sur un automate de reconnaissance.

Compilez le fichier `test_lexeme.c`, soit en utilisant le `Makefile` fourni (avec la commande `make`), soit en utilisant votre environnement de développement (Visual Studio Code, ou autre).

Exécutez ***test_lexeme*** sur la séquence `entree1.txt` : `./test_lexeme entre1.txt`. Vérifiez que vous comprenez ce qui est affiché à l'écran. Est-ce que `entree1.txt` contient une expression arithmétique (simple) correcte ?

Ecrivez (avec un éditeur de votre choix) un fichier ***entree2.txt*** contenant une séquence de lexème "incorrecte" (sur une seule ligne). Vérifiez que les erreurs lexicales sont bien detectées ...

Exercice 2 - compléter l'analyse lexicale

Dessinez l'automate utilisé dans la procédure `reconnaitre_lexeme`.

Complétez (le dessin de) cet automate pour ajouter un opérateur de division (`'/'`).

Intégrez ces extensions dans l'analyseur lexical (en étendant le type `Lexeme` et la procédure `reconnaitre_lexeme`).

(suite page suivante)

Exercice 3 - analyser une expression

Ecrivez maintenant un module ***analyse_syntaxique*** (donc un fichier `analyse_syntaxique.c` et `analyse_syntaxique.h`) qui fournit l'interface suivante :

```
void analyser (char *fichier) ;
   -- e.i : indifferent
   -- e.f : une Expression Arithmetique a ete lue dans fichier
   -- si elle ne contient pas d'erreur de syntaxe un message est affiche
   -- sinon le pgm termine sur un message d'erreur
```

Indication : On rappelle que la syntaxe informelle d'une Expression Arithmetique est définie par l'*expression régulière* suivante :

$$\text{entier}.(\text{opérateur}.\text{entier})^*$$

Vous pouvez donc utiliser l'*automate* vu en cours pour écrire la procédure `analyser()`.

En vous inspirant du programme `test_lexeme.c` :

1. Ecrivez un programme principal `calculette.c` qui appelle la procédure `analyser()` ;
2. le cas échéant complétez le `Makefile` pour compiler ce programme ;
3. testez ce programme sur divers exemples.

Exercice 4 - évaluer une expression

Modifiez maintenant le module ***analyse_syntaxique*** avec cette nouvelle interface :

```
void analyser (char *fichier, int *resultat) ;
   -- e.i : indifferent
   -- e.f : une Expression Arithmetique a ete lue dans fichier
   -- si elle ne contient pas d'erreur sa valeur est dans *resultat
   -- sinon le pgm termine sur un message d'erreur
```

Indication : Il suffit de compléter le code de l'exercice 3 pour que le calcul se fasse de gauche à droite, au fur et à mesure de la lecture de l'expression (on ne tient pas compte des priorités des opérateurs dans cette version !).

Modifiez votre programme principal qui appelle cette procédure pour qu'il affiche le résultat et testez ce programme sur divers exemples.

Extensions

1. Etendez votre calculette pour qu'elle permette d'effectuer des calculs sur des "nombres à virgule".
   Exemple : `2.53 + 0.08 * 2`

2. Enrichir l'analyse lexicale avec les extensions suivantes :
   (a) permettre l'utilisation d'opérateurs représentés par des chaînes de caractères (`plus`, `moins`, etc.). Vous pouvez par exemple utiliser la fonction `strcmp` qui permet de comparer deux chaînes de caractères.
   (b) produire un lexème `ERREUR` en cas d'erreur lexicale sans interompre complètement l'analyse lexicale

A déposer sur Moodle (Avant le 04/02)

Les listings des programmes que vous avez écrits pour ce TP1 et les jeux de test correspondants.